

Reminders

- Sign up for CMPS 182 on piazza. All announcements, lecture notes, homework, labs will be posted there.
- Textbook is required
- Create Gradiane account and join this class, using class code in the Piazza post.
- Gradiane first online assignment
 - “HW 3” message posted on piazza
 - Only 2 questions
 - Due Friday end of day
- Lab assignment 1 has been posted on piazza under “Resources”
 - Due next Friday at 6:00 PM

Basic SQL

Instructor: David Martin

Reference:

*A First Course in Database Systems,
3rd edition, Chapter 2.3, 6.1, 6.2*

Reminder: What is a Data Model?

- A *data model* is a mathematical formalism that specifies:
 1. Structure of the data
 2. Operations on the data
 3. Constraints on the data
- This course focuses mainly on the relational data model
- This lecture gives some basic elements for expressing aspects of (1) and (2)
- **What is the associated query language commonly used for the relational data model?**

SQL – Structured Query Language

- SQL is the principal language used to describe and manipulate data stored in relational database systems.
 - Frequently pronounced as “Sequel”, but formally it’s “Ess Cue El”
 - Not the same as Codd’s relational algebra or relational calculus
 - But largely based on the relational algebra
- Several iterations of the standard from cooperating groups
 - SQL-86, SQL-89, **SQL-92 (SQL2)**, SQL:1999 (SQL3), SQL:2003, SQL:2006, SQL:2008, SQL:2011
 - ANSI (American National Standards Institute)
 - ISO (International Organization for Standards)

SQL DDL and SQL DML

- Two main aspects to SQL:
 - Data Definition Language (DDL)
 - Sublanguage of SQL used to create, delete, modify the definition of tables and views.
 - For declaring database schemas.
 - Data Manipulation Language (DML)
 - Sublanguage of SQL that allows users to insert, delete, and modify rows of tables and pose queries to retrieve data.
 - For asking questions about the database and modifying the database.

Reference:

*A First Course in Database Systems,
3rd edition, Chapter 6.1*

SQL DDL and SQL DML

- Two main aspects to SQL:
 - **Data Definition Language (DDL)**
 - Sublanguage of SQL used to create, delete, modify the definition of tables and views.
 - For declaring database schemas.
 - Data Manipulation Language (DML)
 - Sublanguage of SQL that allows users to insert, delete, and modify rows of tables and pose queries to retrieve data.
 - For asking questions about the database and modifying the database.

Reference:

*A First Course in Database Systems,
3rd edition, Chapter 6.1*

Most of the Primitive Data Types in SQL

- CHAR(n): fixed-length string of up to n characters (blank-padded)
- VARCHAR(n): also a string of up to n characters
- BIT(n)
- BIT VARYING(n)
- BOOLEAN: true/false/unknown
- INT or INTEGER
 - Analogous to int in C
- SHORTINT
 - Analogous to short int in C

Primitive Data Types in SQL (continued)

- DECIMAL(n,d)
 - Total of n decimal digits; d of them to the right of the decimal point
- FLOAT(p), FLOAT and REAL (Implementation-specific)
 - DOUBLE PRECISION
 - Analogous to `double` in C
- DATE and TIME (and TIMESTAMP)
 - See next slide
- A few others ...
- PostgreSQL has non-standard TEXT, for variable strings of any length

Date and Time

- DATE and TIME (and TIMESTAMP)
 - Separate data types
 - Constants are character strings of a specific form
 - DATE '2015-01-13'
 - TIME '16:45:33'
 - TIMESTAMP '2015-01-13 16:45:33'
 - DATE, TIME and TIMESTAMP can be compared using ordinary comparison operators
 - ... WHERE ReleaseDate <= DATE '1990-06-19' ...
 - See Chapter 6.1.5 of Textbook (page 251) for some more details.
 - There are some implementation-specific differences on formats.

Defining a Table

```
CREATE TABLE Movies (  
    title          CHAR(100),  
    year           INT,  
    length         INT,  
    genre          CHAR(10),  
    studioName     CHAR(30),  
    producerC#     INT  
);
```

title	year	length	genre	studioName	producerC#
-------	------	--------	-------	------------	------------

Think of producerC# as the Certificate Number for the movie's producer, where Certificate Number is a key uniquely identifying a Movie Executive.

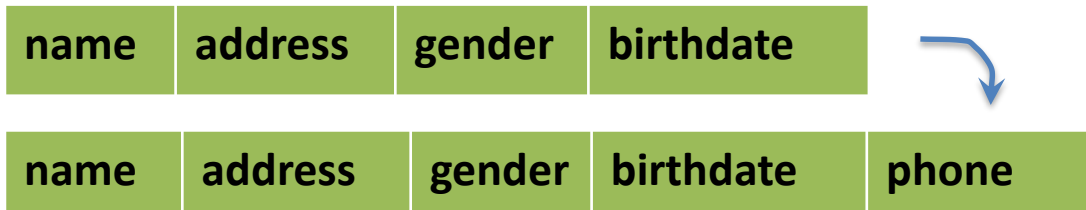
Defining a Table (continued)

```
CREATE TABLE MovieStar (  
    name          CHAR(30),  
    address       VARCHAR(255),  
    gender        CHAR(1),  
    birthdate     DATE  
);
```

name	address	gender	birthdate
------	---------	--------	-----------

Modifying Relation Schemas

- DROP TABLE Movies;
 - The entire table is removed from the database schema.
- ALTER TABLE MovieStar ADD phone CHAR(16);
 - Adds an attribute “phone” with type CHAR(16) to the table MovieStar.



- ALTER TABLE MovieStar DROP birthdate;



Default Values

```
CREATE TABLE MovieStar (  
    name            CHAR(30),  
    address         VARCHAR(255) DEFAULT "Hollywood",  
    gender          CHAR(1),  
    birthdate       DATE  
);
```

- If a new row is inserted and no value is specified for the attribute address, then the value for this attribute will default to "Hollywood".
- If no default value is declared explicitly, then the value of the attribute will default to NULL.
 - NULL is a special value in SQL to represent unknown values.
 - A constraint may prevent a column from having null values.

More Examples on Default Values

```
CREATE TABLE MovieStar (  
    name            CHAR(30),  
    address         VARCHAR(255)  DEFAULT    'Hollywood',  
    gender          CHAR(1)       DEFAULT    '?',  
    birthdate       DATE          DEFAULT    '0000-00-00',  
);
```

```
ALTER TABLE MovieStar ADD phone CHAR(16) DEFAULT 'unlisted';
```

More on NULL

```
CREATE TABLE MovieStar (  
    name            CHAR(30)  PRIMARY KEY,  
    address         VARCHAR(255) NOT NULL DEFAULT "Hollywood",  
    gender          CHAR(1),  
    birthdate       DATE NOT NULL  
);
```

- If no default value is declared explicitly and no value is entered for an attribute, then the value of the attribute will default to NULL.
 - NULL is a special value in SQL to represent unknown values.
Also used when there is no applicable value, or a value is withheld for security / privacy reasons.
 - **NOT NULL** constraint prevents a column from having null values.
 - Attributes in the PRIMARY KEY can't be null.

Some Facts About Nulls

- Almost all comparisons with NULL will evaluate to unknown. If Salary is NULL, then the following will be UNKNOWN:
 - Salary = 10
 - Salary <> 10
 - 90 > Salary OR 90 <= Salary
 - Salary = NULL (!)
 - Salary <> NULL (!)
- Use of IS NULL and NOT IS NULL
 - Salary IS NULL will be true if Salary is NULL, false otherwise
 - Salary IS NOT NULL will be true if Salary isn't NULL, false otherwise
- ORDER BY works with attributes that can have NULL values
 - NULL will probably be smallest or largest value
 - Not specified by SQL standard, so it depends on the implementation

Some More Examples For Nulls

- If Salary1 has value NULL and Salary2 has value NULL, then what will be the value for these predicates?
 - Salary1 = Salary2
 - Salary1 <> Salary2
 - Salary1 IS NULL
 - Salary2 IS NOT NULL
 - Salary1 IS NULL OR Salary2 IS NOT NULL
- Write a condition that allows the names of students whose Major is either Computer Science or NULL.
- Write a condition that allows the names of students whose Major isn't NULL and whose avgGPA is NULL.

Declaring Keys

Two ways to declare a single attribute to be a key in the CREATE TABLE statement:

```
CREATE TABLE MovieStar (  
    name          CHAR(30) PRIMARY KEY,  
    address       VARCHAR(255),  
    gender        CHAR(1),  
    birthdate     DATE  
);
```

```
CREATE TABLE MovieStar (  
    name          CHAR(30),  
    address       VARCHAR(255),  
    gender        CHAR(1),  
    birthdate     DATE,  
    PRIMARY KEY (name)  
);
```

Declaring Keys (continued)

If the key consists of multiple attributes, then those attributes can be declared as a key as follows:

```
CREATE TABLE Movies (  
    title      CHAR(100),  
    year       INT,  
    length     INT,  
    genre      CHAR(10),  
    studioName CHAR(30),  
    producerC# INT,  
    PRIMARY KEY (title,year)  
);
```

PRIMARY KEY vs. UNIQUE

- In the previous examples, the keyword “PRIMARY KEY” can be replaced by “UNIQUE”.
 - Both specify that attributes are keys, but PRIMARY KEY is **not** identical to UNIQUE.

```
CREATE TABLE MovieStar (  
    name        CHAR(30),  
    address     VARCHAR(255),  
    gender      CHAR(1),  
    birthdate   DATE,  
    PRIMARY KEY (name)  
);
```

```
CREATE TABLE MovieStar (  
    name        CHAR(30),  
    address     VARCHAR(255),  
    gender      CHAR(1),  
    birthdate   DATE,  
    UNIQUE (name)  
);
```

- In the standard, SQL Tables aren't required to have a key (primary or otherwise), but some implementations require it (or create it).

PRIMARY KEY vs. UNIQUE (continued)

```
CREATE TABLE MovieStar (  
    name          CHAR(30),  
    address       VARCHAR(255),  
    gender        CHAR(1),  
    birthdate     DATE,  
    PRIMARY KEY (name)  
);
```

- None of the rows in MovieStar can have null *name* values.
- Rows are uniquely identified by their *name* values.
- There can be at most one primary key for a table.

```
CREATE TABLE MovieStar (  
    name          CHAR(30),  
    address       VARCHAR(255),  
    gender        CHAR(1),  
    birthdate     DATE,  
    UNIQUE (name)  
);
```

- Rows in MovieStar can contain null *name* values.
- Rows in MovieStar with non-null *name* values are uniquely identified by their *name* values.
- There can be multiple unique constraints for a table in addition to a primary key.

SQL DDL and SQL DML

- Two main aspects to SQL:
 - Data Definition Language (DDL)
 - Sublanguage of SQL used to create, delete, modify the definition of tables and views.
 - For declaring database schemas.
 - Data Manipulation Language (DML)
 - Sublanguage of SQL that allows users to insert, delete, and modify rows of tables and pose **queries** to retrieve data.
 - For asking questions about the database and modifying the database.

Reference:

*A First Course in Database Systems,
3rd edition, Chapter 6.1*

Database Schema for Our Running Example

- Let's assume we have a database schema with five relation schemas.

Movies(title, year, length, genre, studioName, producerC#)

StarsIn(movieTitle, movieYear, starName)

MovieStar(name, address, gender, birthdate)

MovieExec(name, address, cert#, netWorth)

Studio(name, address, presC#)

*In this schema, **cert#** is just an attribute for a MovieExec's "certificate number"; producerC# and presC# should refer to the cert# of the producer of a Movie and president of a Studio, respectively. Nothing special about these attribute names for SQL—it's just an example.*

A Simple SQL Query

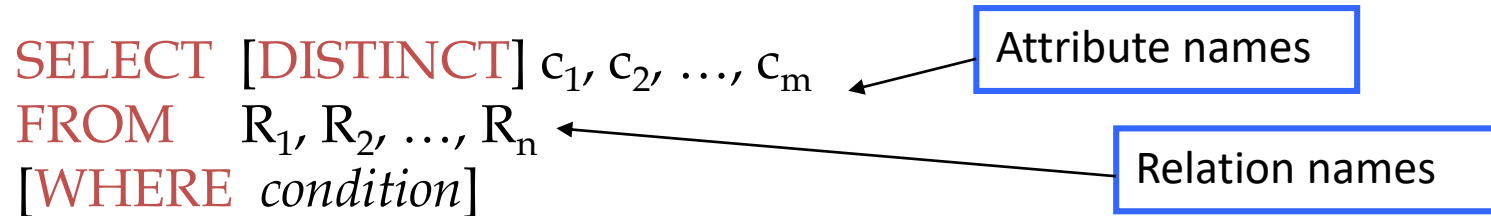
- Find all movies produced by Disney Studios in 1990.

```
SELECT      *  
FROM Movies  
WHERE studioName = 'Disney' AND year = 1990;
```


Basic Form of a SQL Query

- Basic form:

`SELECT` [`DISTINCT`] c_1, c_2, \dots, c_m ← Attribute names
`FROM` R_1, R_2, \dots, R_n ← Relation names
[`WHERE` *condition*]

The diagram illustrates the basic form of an SQL query. It shows the keywords 'SELECT', 'FROM', and 'WHERE' in red. The optional keyword 'DISTINCT' is in brackets. The attribute names c_1, c_2, \dots, c_m are followed by an arrow pointing to a blue box labeled 'Attribute names'. The relation names R_1, R_2, \dots, R_n are followed by an arrow pointing to a blue box labeled 'Relation names'. The 'WHERE' clause is followed by the word 'condition' in italics.

- We will focus on one relation R_1 for now.

A Simple SQL Query

- Find all movies produced by Disney Studios in 1990.

SELECT *

FROM Movies

WHERE studioName = 'Disney' AND year = 1990;

The symbol “*” is a shorthand for all attributes of relations in the FROM clause.

Movies

Title	Year	Length	Genre	studioName	producerC#
Pretty Woman	1990	119	Romantic	Disney	999
Monster's Inc.	1990	121	Animation	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

A Simple SQL Query (Result)

- Find all movies produced by Disney Studios in 1990.

```
SELECT *
```

```
FROM Movies
```

```
WHERE studioName = 'Disney' AND year = 1990;
```

Result

Title	Year	Length	Genre	studioName	producerC#
Pretty Woman	1990	119	true	Disney	999

An Even Simpler SQL Query

- Find all movies.

```
SELECT *  
FROM Movies;
```

Equivalent to:

```
SELECT *  
FROM Movies  
WHERE true;
```

Movies	Title	Year	Length	Genre	studioName	producerC#
	Pretty Woman	1990	119	Romantic	Disney	999
	Monster's Inc.	1990	121	Animation	Dreamworks	223
	Jurassic Park	1998	145	NULL	Disney	675

An Even Simpler SQL Query (Result)

- Find all movies.

```
SELECT *
```

```
FROM Movies;
```

Result

Title	Year	Length	Genre	studioName	producerC#
Pretty Woman	1990	119	Romantic	Disney	999
Monster's Inc.	1990	121	Animation	Dreamworks	223
Jurassic Park	1998	145	NULL	Disney	675

2 Fundamental Concepts

- *Projection*: retrieve a subset of the *columns* of a table
- *Selection*: retrieve a subset of the *rows* of a table
- As we will see before long, these concepts are formalized in relational algebra.
- They are also commonly understood in the database world and used informally.
- They are also present in SQL ...

A Simple SQL Query with Projection

- Return the title and year of all movies.

```
SELECT title, year  
FROM Movies;
```

Projection:

- Only a subset of *columns* from the table(s) in the FROM clause is to be retrieved.

Result

Title	Year
Pretty Woman	1990
Monster's Inc.	1990
Jurassic Park	1998

A Simple SQL Query with Projection and Selection

- Return the title and year of all movies produced by Disney Studios in 1990.

SELECT title, year

FROM Movies

WHERE studioName = 'Disney' AND year = 1990;

Result

Title	Year
Pretty Woman	1990

Selection:

- Only a subset of *rows* from the table(s) in the FROM clause is to be retrieved.

Sets vs. Bags (or Multisets)

From basic set theory –

- Every element in a set is distinct
 - E.g., $\{2,4,6\}$ is a set but $\{2,4,6,2,2\}$ is not a set.
- A bag (or multiset) may contain repeated elements.
 - E.g., $\{\{2,4,6\}\}$ is a bag. So is $\{\{2,4,6,2,2\}\}$.
 - Note that double set brackets in $\{\{2,4,6\}\}$ indicate it's a bag, not a set
 - Equivalently written as $\{\{2[3],4[1],6[1]\}\}$.
- The order among elements in a set or bag is not important
 - E.g., $\{2,4,6\} = \{4,2,6\} = \{6,4,2\}$
 - $\{\{2,4,6,2,2\}\} = \{\{2,2,2,6,4\}\} = \{\{6,2,2,4,2\}\}$.

DISTINCT: Sets vs Multisets/Bags

- Return the years of all movies with length less than 140.

```
SELECT year  
FROM Movies  
WHERE length < 140;
```

Result

Year
1990
1990

Multiset or bag semantics

```
SELECT DISTINCT year  
FROM Movies  
WHERE length < 140;
```

Result

Year
1990

Set semantics

Note: this means rows of the *results* table are *distinct*

DISTINCT: Sets vs Multisets/Bags

- Return the years of all movies with length less than 140.

```
SELECT title, year  
FROM Movies  
WHERE length < 140;
```

Result

Title	Year
Pretty Woman	1990
Monsters, Inc.	1990

Multiset or bag semantics

```
SELECT DISTINCT title, year  
FROM Movies  
WHERE length < 140;
```

Result

Title	Year
Pretty Woman	1990
Monsters, Inc.	1990

Set semantics

Here we keep both rows in the results table, because with these 2 attributes the rows are distinct

Aliasing Attributes

- Allows you to rename the attributes of the result.
- Example: Return the title and length of all movies as attributes name and duration.

```
SELECT title AS name, length AS duration  
FROM Movies;
```

Result

name	duration
Pretty Woman	119
Monster's Inc.	121
Jurassic Park	145

Expressions in the SELECT Clause

- Expressions are allowed in the SELECT clause.
- Return the title and length of all movies as name and duration in hours (durationInHours)

```
SELECT title AS name, length * 0.016667 AS durationInHours  
FROM Movies;
```

Result

name	durationInHours
Pretty Woman	1.98
Monster's Inc.	2.02
Jurassic Park	2.42

Constants in the Result

- Constants can also be included in the SELECT clause.
- Every row will have the same constant specified in the SELECT clause.

```
SELECT title AS name, length * 0.016667 AS durationInHours,  
       'hrs.' as inHours  
FROM Movies;
```

Result

name	durationInHours	inHours
Pretty Woman	1.98	hrs.
Monster's Inc.	2.02	hrs.
Jurassic Park	2.42	hrs.

More on the Conditions in the WHERE Clause

- WHERE studioName = 'Disney' AND year = 1990;
- Comparison operators:
 - =, <>, <, >, <=, >=
 - Equal, not equal, less than, greater than, less than or equal, greater than or equal
 - E.g., WHERE year <= 1990
- Logical connectives:
 - AND, OR, NOT
 - E.g., WHERE NOT (studioName = 'DISNEY' AND year <= 1990)
- Arithmetic expressions:
 - +, -, *, /, etc.
 - E.g., WHERE ((length*0.01667) > 2 OR (length < 100)) AND year > 2000

More on the Conditions in the WHERE Clause (cont'd)

- In general, the WHERE clause consists of a boolean combination of conditions using logical connectives AND, OR, and NOT.
- Each condition is of the form
expression op expression
 - *expression* is a column name, a constant, or an arithmetic or string expression
 - *op* is a comparison operator (=, <>, <, >, <=, >=, LIKE)

(More on this soon)



String Comparisons

- Strings are compared in lexicographical order.
 - Let $a_1.a_2. \dots .a_n$ and $b_1.b_2. \dots b_m$ be two strings.
 - Then $a_1.a_2. \dots .a_n < b_1.b_2. \dots b_m$ if either:
 1. $a_1.a_2. \dots .a_n$ is a proper prefix of $b_1.b_2. \dots b_m$, or,
 2. For some $1 \leq i < n$, we have $a_1=b_1, a_2=b_2, \dots, a_{i-1}=b_{i-1}$ and $a_i < b_i$.
- Examples:
 - ‘Pretty’ < ‘Pretty Woman’,
 - ‘butterfingers’ < ‘butterfly’

Pattern Matching in SQL

- LIKE operator
 - s LIKE p, s NOT LIKE p
 - s is a string, p is a pattern
- % (stands for 0 or more arbitrary chars)
- _ (stands for exactly one arbitrary char)

SELECT *

FROM Movies

WHERE title LIKE 'Pretty%'

title LIKE 'P_etty%'

Pattern Matching in SQL

- How do you match titles with an apostrophe?

- E.g., *Monster's Inc.*

```
SELECT *
```

```
FROM Movies
```

```
WHERE title LIKE 'Monster's Inc.'
```

- **Wrong!!!**

- Correct:

```
WHERE title LIKE 'Monster"s Inc.'
```

- How would you match the string containing exactly 2 apostrophes?

```
WHERE title LIKE ''''
```

SQL's Three-Valued Logic

Students	studentID	name	major	gender	avgGPA
	112	Ann	Computer Science	F	NULL
	327	Bob	NULL	M	3.90
	835	Carl	Physics	M	4.00

What rows are selected by the condition
major = 'Computer Science' AND avgGPA > 3

- Three-valued logic TRUE, FALSE, UNKNOWN
 - NULL is synonymous with UNKNOWN.
- If major is NULL, then **major='Computer Science'** evaluates to UNKNOWN.
- If avgGPA is NULL, then **avgGPA > 3** evaluates to UNKNOWN.
- If the condition evaluates to UNKNOWN, then the tuple is *not* returned.

SQL's Three-Valued Logic: Truth Table

p	q	$p \text{ OR } q$	$p \text{ AND } q$	$p = q$
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

p	NOT p
True	False
False	True
Unknown	Unknown

Ordering the Result

```
SELECT [DISTINCT] <list of attributes>  
FROM   R1, R2, ..., Rn  
[WHERE condition]  
[ORDER BY <list of attributes>]
```

- ORDER BY presents the result in a sorted order.
- By default, the result will be ordered in ascending order.
For descending order, you write:
 - ORDER BY <list of attributes> DESC

ORDER BY

SELECT *

FROM Movies

WHERE year > 1980

ORDER BY length, title;

- The result will list movies that satisfy the condition in the WHERE clause in ascending order of length, then by title.
 - Shortest length movies will be listed first.
 - Among all movies with the same length, the movies will be sorted in ascending order of title.
- **ORDER BY length DESC, title;**
 - Longest length movies will be listed first.
 - Among all movies with the same length, the movies will be sorted in ascending order of title.

More on ORDER BY

- You can ORDER BY expressions, not just attributes
 - ORDER BY Quantity * Price
 - ORDER BY Quantity * Price DESC
- ORDER BY works with attributes that can have NULL values
 - NULL will probably be smallest or largest value
 - Not specified by SQL standard, so it depends on the implementation

Meaning of an SQL Query with One Relation in the FROM Clause

```
SELECT [DISTINCT]  $c_1, c_2, \dots, c_m$   
FROM  $R_1$   
[WHERE condition]  
[ORDER BY <list of attributes>] [DESC]
```

- Let Result denote an empty collection.
- For every tuple t from R_1 ,
 - if t satisfies *condition* (i.e., if condition evaluates to true), then add the tuple that consists of c_1, c_2, \dots, c_m components of t into Result.
- If DISTINCT is in the SELECT clause, then remove duplicates in Result.
- If ORDER BY <list of attributes> exists, then order the tuples in Result according to ORDER BY clause.
- Return Result.